CS271 Computer Graphics II

Programming Assignment 5

Supplementary Notes

Spring 2021

Programming Assignment 5

- Tasks:

1. Compute **normal** at each vertex and render
2. Compute **mean curvature** at each vertex and render
3. Implement Explicit Laplacian smoothing
   - using uniform weights and cotangent weights
4. Implement Implicit Laplacian smoothing
   - using uniform weights and cotangent weights

Normal at a vertex

- [Siggraph 2000 subdivision course] The normal vector at an interior vertex of valence k can be computed as

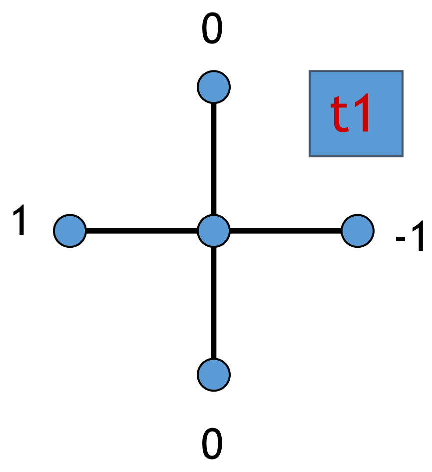
$$t_1 \times t_2,$$

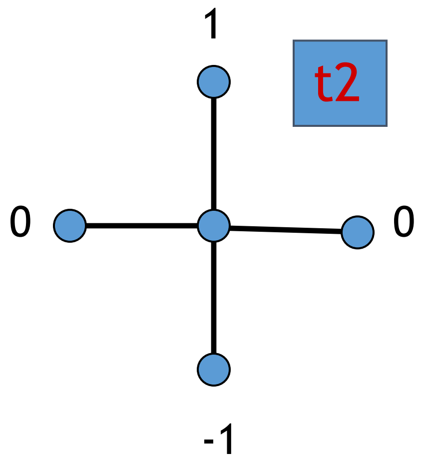
where t_i are tangent vectors computed as:

$$\boldsymbol{t}_1 = \sum_{i=0}^{k-1} \cos \frac{2\pi i}{k} \boldsymbol{p}_i$$

$$\boldsymbol{t}_2 = \sum_{i=0}^{k-1} \sin \frac{2\pi i}{k} \boldsymbol{p}_i$$

Masks for valence four





Normal at a vertex

- At a boundary vertex $\boldsymbol{p}$ with valence k, the normal can be computed as

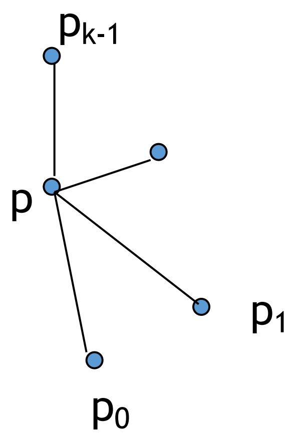
$$\boldsymbol{t}_{along} \times \boldsymbol{t}_{across}$$

where

$$\boldsymbol{t}_{along} = \boldsymbol{p}_0 - \boldsymbol{p}_{k-1}$$

$$\boldsymbol{t}_{across} = \begin{cases} \boldsymbol{p}_0 + \boldsymbol{p}_1 - 2\boldsymbol{p} & k = 2 \\ \boldsymbol{p}_1 - \boldsymbol{p} & k = 3 \\ \sin\theta(\boldsymbol{p}_0 + \boldsymbol{p}_{k-1}) + (2\cos\theta - 2)\sum_{i=1}^{k-2} \sin i\theta\ \boldsymbol{p}_i & k \geq 4 \end{cases}$$

$$\textit{where} \quad \theta = \pi/(k-1)$$



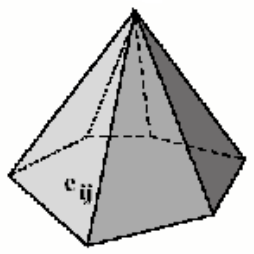
Vertex normals

- You can store the normal of a vertex in Vertex::normal by using the function Vertex::SetNormal
- The normals will be used for rendering in smooth shading mode
- You should re-compute the normal after applying smoothing

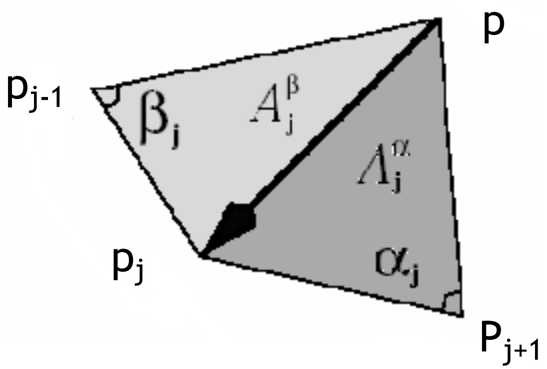
Mean curvature at a vertex

- [Siggraph99 Desbrun et al] The discrete mean curvature at an interior vertex **p** with valence k can be computed as the L2-norm of

$$\bar{\kappa}\boldsymbol{n} = -\frac{1}{4A}\sum_{j=0}^{k-1}(\cot\alpha_j + \cot\beta_j)(\boldsymbol{p}_j - \boldsymbol{p})$$



where A is the sum of areas of all the triangles sharing the vertex p



Rendering in OpenGL (Flat)

上海科技大学 ShanghaiTech University

Flat shaded

```
glShadeModel(GL_FLAT);
glEnable(GL_LIGHTING);
...
glBegin(GL_TRIANGLES);
for(....) {
    ....
    glNormal3fv(n);
    glVertex3fv(v1);
    glVertex3fv(v2);
    glVertex3fv(v3);
}
glEnd();
```

One normal for a triangle

立志成才 报国裕民

Rendering in OpenGL (Smooth)

Smooth shaded

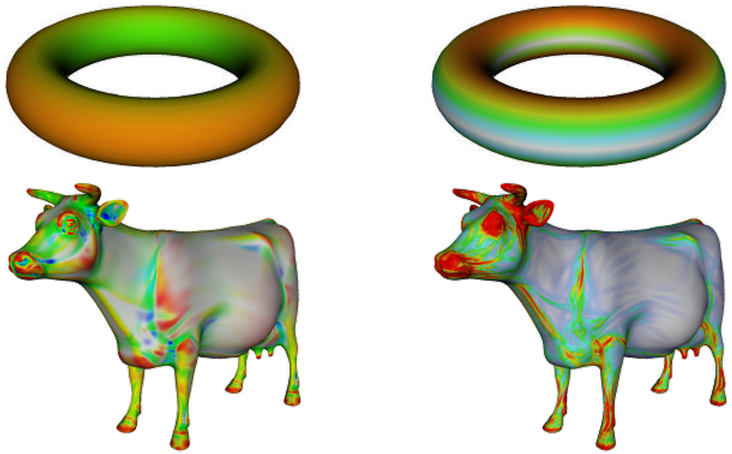
```
glShadeModel(GL_SMOOTH);
glEnable(GL_LIGHTING);
...
glBegin(GL_TRIANGLES);
for(....) {
    ....
    glNormal3fv(n1);
    glVertex3fv(v1);
    glNormal3fv(n2);
    glVertex3fv(v2);
    glNormal3fv(n3);
    glVertex3fv(v3);
    ....
}
glEnd();
```

One normal for each vertex

Note: for very coarse models, smooth shading may still look faceted.

Render the curvatures

- Using any smooth color ramp

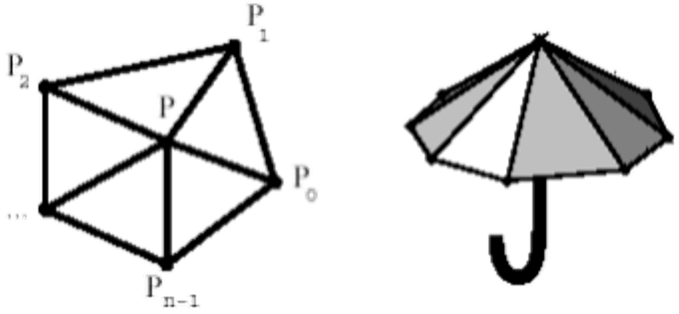


- Render using DrawColorSmoothShaded();

Explicit Laplacian Smoothing

- Fairing operator (uniform)

$$\Delta x_i = \frac{1}{k}\sum_{j=0}^{k-1} x_j - x_i \qquad \boldsymbol{x}_i^{new} \leftarrow \boldsymbol{x}_i + \lambda \Delta \boldsymbol{x}_i$$



- In matrix form

$$\mathbf{X}_{t+1} = \mathbf{X}_t + \lambda \mathbf{L}(\mathbf{X}_t)$$

$$\Rightarrow \quad \mathbf{X}_{t+1} = (\mathbf{I} + \lambda \mathbf{L})\mathbf{X}_t$$

Drawbacks: small time step for large mesh → slow

Uniform Laplacian

where t is time stamp

Explicit Laplacian Smoothing

上海科技大学 ShanghaiTech University

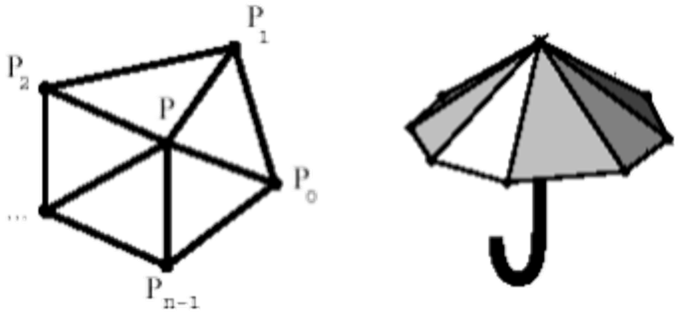
- Instead of uniform-weight Laplacian, use cotangent-weight Laplacian

$$\Delta x_i = \frac{1}{\sum w_{ij}} \sum_{j=0}^{k-1} w_{ij} x_j - x_i,$$

$$w_{ij} = \cot \alpha_{ij} + \cot \beta_{ij}$$

$$\boldsymbol{x}_i^{new} \leftarrow \boldsymbol{x}_i + \lambda \Delta \boldsymbol{x}_i$$

$$\mathbf{X_{t+1}} = \mathbf{X_t} + \lambda \mathbf{L}(\mathbf{X_t})$$



立志成才报国裕民

Explicit Laplacian Smoothing

上海科技大学 ShanghaiTech University

- The matrix element of L
  - L is sparse and has the following elements:

$$L_{ij} = \begin{cases} -1 & if\ i = j \\ \dfrac{1}{\sum_j w_{ij}} w_{ij}, & if\ (i,j) \in E \\ 0 & otherwise \end{cases}$$

  - Use the Matrix::AddElement() to build up the matrix.

立志成才报国裕民

Implicit Laplacian Smoothing

- **Implicit** updating

$$\mathbf{X}_{\mathbf{t+1}} = \mathbf{X}_{\mathbf{t}} + \lambda \mathbf{L}(\mathbf{X}_{\mathbf{t+1}})$$

$$\Rightarrow (\mathbf{I} - \lambda \mathbf{L})\mathbf{X}_{\mathbf{t+1}} = \mathbf{X}_{\mathbf{t}}$$

You need to solve a
sparse linear system

- Allows large time step
- In your implementation setting λ to 1 is okay
- Solve the linear system using **biconjugate gradient** (BCG) method or **conjugate gradient** method (see reference)

Implementation Hints

上海科技大学
ShanghaiTech University

- Using the class Matrix in the sample code.
  - Interface provided to you:
    - Matrix::**AddElement**(int row, int col, double value);
    - Matrix::**Multiply**(double* xIn, double* xOut);
    - Matrix::**PreMultiply**(double* xIn, double* xOut);
  - Interface you need implement:
    - Matrix::**BCG**(double* b, double* x, int maxIter, double tolerance);

Implement the BCG solver here

立志成才报国裕民

Implementation Hints

- Construct the linear solver
  - Please refer to the handout on painless-conjugate-gradient.
  - Specifically, you can directly go to page 32 for implementation details.